REMARKS

Claims 1-53 are pending in the present application. Claims 51-53 have been added herewith. Reconsideration of the claims is respectfully requested.

I. 35 U.S.C. § 102, Anticipation

The Examiner rejected Claims 1, 22 and 43 under 35 U.S.C. § 102 as being anticipated by Bogda et al., "Removing Unnecessary Synchronization in Java", Technical Report TRCS99-10, University of California, Dept. of Computer Science, April 1999. This rejection is respectfully traversed.

Generally speaking, the present invention is directed to a method, system and computer program product for *detecting resource exception errors*. The cited reference teaches a method for *removing unnecessary synchronization in Java code*. While such reference alludes to accommodating preexisting exception handling code (which uses traditional Java throw and catch statements), that is the extent of such exception handling – that the described routine accommodates preexisting exception handling code that includes throw and catch statements. However, the teachings of Bodga *do not detect resource exception errors*, as expressly claimed in Claim 1, and in particular the cited Bodga reference does not teach the claimed three-step synergistic process for detecting resource exception errors. At best, the cited Bodga reference detects pre-existing throw and catch statements (that already exists in the code to perform pre-existing exception handling) when converting Java bytecodes to an internal canonical representation as part of the overall process of removing unnecessary synchronization from such Java code.

Specifically with respect to Claim 1, such claim recites a method for detecting resource exception errors, and as a part of such resource exception error detection recites steps of (1) scanning a code for a first method invocation used to open a first resource file; (2) identifying said first method invocation; and (3) opening said first resource file using said first method invocation to detect resource exception errors. As can be seen, each subsequent step in Claim 1 synergistically co-acts with a preceding step to provide an overall ability to *detect resource exception errors*. Code is scanned for a first method invocation used to open a first resource file. Such first method invocation is identified.

This first method invocation is used when opening the first resource file, the first resource file being opened to detect resource exception errors. As can be seen, the claimed first method invocation and the claimed first resource file are involved in several aspects of the resource exception error detection methodology of Claim 1. Code is scanned for a first method invocation *used to open* such first resource file. This first method invocation is used to open such first resource file in order *to detect resource exception errors*. The cited reference does not teach or otherwise suggest such a first resource file that is opened, using a first method invocation, to detect resource exception errors, as will now be described in detail.

In rejecting Claim 1, the Examiner cites two separate and disjoint sections of the cited Bodga reference. As to the overall claimed process of detecting resource exception errors, the Examiner cites the following paragraph at Bodga page 5:

> "Some Java constructs do not map trivially to this form and require special attention. To accommodate exception handling, we treat throw and catch statements as static field accesses. Assuming the class Exception has a static field named escapes, we view statements of the form throw x as Exception.escapes = x and statements of the form catch x as x = Exception.escapes. Similarly, since the compiler cannot generally distinguish two array elements, we view all array cell accesses as accesses to the single fictitious array instance field. The statement a[i] = x thus transforms to a.array = x."

As can be seen, this passage describes how to accommodate pre-existing throw and catch statements that are a part of the pre-existing code being scanned, such throw and catch statements being used for handling exceptions. *In order to accommodate such pre-existing exception handing, the throw and catch statements are treated as static field accesses*. That is the extent of Bodga's discussion of exception handling – that throw and catch statements are treated as static field accesses. As a part of such Bodga exception handling, there is *no* teaching of (1) scanning a code for a first method invocation used to open a first resource file; or (2) opening said first resource file using said first method

invocation to detect resource exception errors. As to missing claimed step (1), the Examiner states that this step is taught by Bodga at the following paragraph on page 14:

> "Some authors have proposed eliminating synchronization in single-threaded programs by detecting that no second thread is constructed [M+97][B97][F+98]. *After scanning an entire program and not finding any calls to Thread constructors*, a compiler can conclude that the program is single-threaded and remove all synchronizations. Similarly, a run-time system could omit synchronization until the second thread is created. Unfortunately, this approach will not succeed in general. In particular, it does not improve multi-threaded programs, such as GUI-based applications. Also, the JDK 1.2 system classes spawn helper threads at the start of every application, making all programs multi-threaded." (emphasis added)

As can be seen, this passage describes a technique for eliminating synchronization in single-threaded programs and has nothing to do with detecting resource exception errors. Rather, a program is scanned and if *no* calls to Thread constructors are found, it is concluded that the program is single threaded and thus all synchronizations can be removed. This passage does *not* teach scanning a code for *a first method invocation used to open a first resource file*. Rather, it teaches scanning code for Thread constructors. There is no teaching of any type of first resource file, or scanning of code for a method invocation that is used to open such (missing) first resource file. For a prior art reference to anticipate in terms of 35 U.S.C. 102, every element of the claimed invention must be identically shown in a single reference. *In re Bond*, 910 F.2d 831, 15 USPQ2d 1566 (Fed. Cir. 1990). Because this cited passage does not teach any type of step involved in detecting resource exception errors, and in particular does not teach a first resource file (which is different from the claimed 'code') or *scanning code for a method invocation used to open such a resource file*, it is shown that Claim 1 is not anticipated by the cited reference as there is at least missing claimed element not taught by the cited reference.

As to missing claimed step (2), the Examiner states that this step is taught by Bodga at the following paragraph on page 5:

> "Some Java constructs do not map trivially to this form and require special attention. *To accommodate exception handling, we treat throw and catch statements as static field accesses.* Assuming the class Exception has a static field named escapes, we view statements of the form throw x as Exception.escapes = x and statements of the form catch x as x = Exception.escapes. Similarly, since the compiler cannot generally distinguish two array elements, we view all array cell accesses as accesses to the single fictitious array instance field. The statement a[i] = x thus transforms to a.array = x." (emphasis added)

with the Examiner also equating 'escapes' or 'Exception.escapes' with the claimed first resource file. Applicants urge that Bodga's 'escapes' and 'Exception.escapes' are temporary variables used to maintain the throw and catch statement parameters as a part of transforming Java bytecode into canonical form (page 5, first paragraph, "Here, x, y.... regarding similar conversion"). The 'escapes' and 'Exceptions.escapes' are not resource files that are opened (but rather are internal variables), and in addition they are not opened *using the first method invocation*, as expressly recited in Claim 1. Thus, Claim 1 is further shown to not be anticipated by the cited reference, as there is yet another missing claimed feature not identically shown in a single reference.

It should also be noted that the above listed passages cited by the Examiner in rejecting Claim 1 are separate and disjoint passages that do not synergistically co-act with one another. The passage cited on page 5 of Bodga deals with transforming Java bytecodes to a canonical form, and specifically with a technique for transforming throw and catch statements in Java bytecode to an internal static field. The passage cited on page 14 deals with scanning code to locate Thread constructors, and these Thread constructors have nothing to do with throw and catch statements, nor do they have anything to do with exception handling or detecting resource exception errors. Instead, detection of Thread constructors is used to determine if synchronizations can be removed

from the code being scanned (Bodga page 14, 1st full paragraph). Bodga's synchronization allows multiple threads to access shared data structures safely (Bodga page 1), and is not part of any type of resource exception error detection. Thus, it is shown that these two disjoint passages of Bodga do not synergistically co-act with one another to provide any type of resource exception error detection, and thus Claim 1 is further shown to not be anticipated by the cited reference, as every element of the claimed invention is not identically shown in a single reference (the synergistic co-action between the three recited steps as a part of the resource exception error detection). Claim 1 expressly recites scanning a code for *a first method invocation* used to open a first resource file; identifying *said first method invocation*; and (3) opening said first resource file *using said first method invocation* to detect resource exception errors, and thus the first method invocation is scanned for, identified, and used to open the resource file.

To summarize, the cited reference does not teach "scanning a code for a first method invocation *used to open a first resource file*" (emphasis added). Rather, the passage cited by the Examiner in rejecting this step is directed to treating throw and catch statements (which are preexisting statements in the Java code used for exception handling) as static field accesses as a part of converting Java bytecodes to an internal canonical form. The cited reference also does not teach using such (missing) first method invocation to open a first resource file. Rather, the passage cited by the Examiner teaches scanning code for Thread constructors as a part of determining whether the code being scanned is single-threaded and therefore can have its synchronization removed. Finally, Bodga's teaching of treating catch and throw statements as static field accesses and Bodga's teaching of scanning code for Thread constructors are separate and unrelated functions do not synergistically co-act with one another (one is directed to converting bytecodes to an internal canonical form, and the other is with respect to detecting whether the code is single-thread), whereas Claim 1 expressly recites scanning a code for *a first method invocation* used to open a first resource file; identifying *said first method invocation*; and (3) opening said first resource file *using said first method invocation* to detect resource exception errors, and thus the first method invocation that is scanned for, identified, and used to open the resource file, and thus this first method invocation links and provides the synergistic co-action between the steps of Claim 1 in order to

advantageously provide an overall method for detecting resource exception errors. Applicants thus urge that Claim 1 has been erroneously rejected under 35 U.S.C. § 102.

Applicants traverse the rejection of Claims 22 and 43 for similar reasons to those given above with respect to Claim 1.

Therefore, the rejection of Claims 1, 22 and 43 under 35 U.S.C. § 102 has been overcome.

II. Objection to Claims

The Examiner stated that Claims 2-21, 23-42 and 44-50 were objected to as being dependent upon a rejected base claim, but would be allowable if rewritten in independent form including all of the limitations of the base claim and any intervening claims. While Applicants graciously acknowledge the allowance of such claims, it is urged that such claims are allowable in their present form as depending or ultimately depending on Claims 1, 22 and 43 (which have been shown above to be allowable over the cited refcrence).

III. Newly Added Claims

Claims 51-53 have been added herewith. Examination of such claims is requested.

IV. Conclusion

It is respectfully urged that the subject application is patentable over the cited reference and is now in condition for allowance. The Examiner is invited to call the undersigned at the below-listed telephone number if in the opinion of the examiner such a telephone conference would expedite or aid the prosecution and examination of this application.

DATE: 3/31/2005

Respectfully submitted,

Duke W. Yee
Reg. No. 34,285
Wayne P. Bailey
Reg. No. 34, 289
Yee & Associates, P.C.
P.O. Box 802333
Dallas, TX 75380
(972) 385-8777
Attorneys for Applicant